Part-1:

- **Arrays:** Concepts, Using Array in C, Array Application, Two Dimensional Arrays, Multidimensional Arrays, Programming Example

Part-2:

- **Strings**: String Concepts, C String, String Input / Output Functions, Arrays of Strings, String Manipulation Functions, Programming Example
- **Enumerated, Structure, and Union:** The Type Definition (Type def), Enumerated Types, Structure, Unions, and Programming Application.

Arrays

Why do we need ARRAYS?

Problem: When we have many data elements (10,20,30…n), we need many different variables (v1,v2,v3…vn). As the number of variables increases, the complexity of the program also increases

Solution: Arrays are used **to store multiple elements in a single variable (v[100]**, instead of declaring separate variables for each value.

Define an ARRAY:

Array is a collection of data elements with a similar data type. They are stored in the contiguous memory location. In the array, the first element is stored in index 0; the second element is stored in index 1, and so on. Arrays can be of a single dimension or multi-dimension.

An array is a special variable that is used to store multiple values of similar data types (homogeneous) at contiguous memory locations.

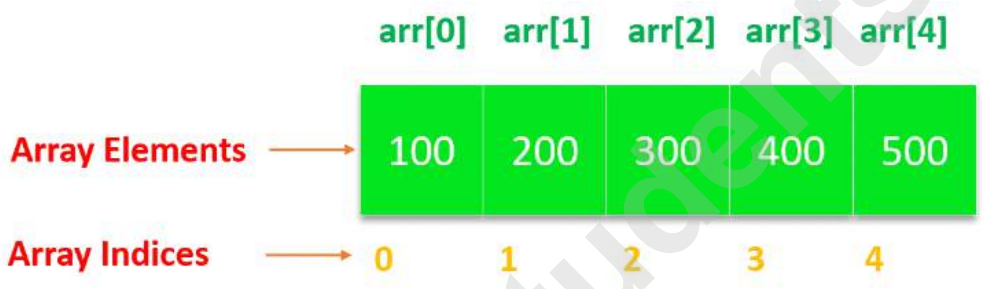
In C programming language, arrays are classified into two types. They are as follows:

- Single-Dimensional Array / One-Dimensional Array
- Multi-Dimensional Array

What are Single-Dimensional or One-Dimensional Arrays?

1. Description of Single Dimensional Array:

Single-dimensional array or 1-D array is the simplest form of array in C. This type of array consists of elements of similar types and these elements can be accessed through their indices (positions).



2. Declaration of Single Dimensional Array:

In C programming language, when we want to create an array we must know

- the datatype of values to be stored in that array and
- also the number of values to be stored in that array.

Syntax1: Create 1-D array with Size:

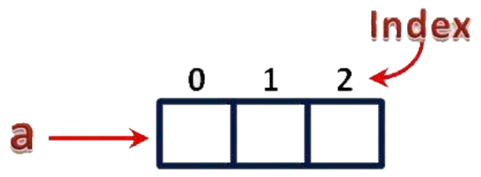
datatype arrayName [size] ;

- **datatype:** data type of array, Example: int, char, float, etc.
- **array_name:** Name of the array.
- **size:** Size of each dimension of the array

Example1: Declaration of 1-D Array with Size

```
// declare an array by specifying size in [].
int a[3];
```

Here, the compiler allocates 12 bytes of contiguous memory locations with a single name 'a' and tells the compiler to store three different integer values (each in 4 bytes of memory) into that 12 bytes of memory. For the above declaration, the memory is organized as follows.



All three memory locations in the above memory allocation have a common name 'a'. So accessing individual memory locations is not possible directly. Hence, the compiler assigns a numerical reference value to every individual memory location of an array. This reference number is called "Index" or "Subscript" or "Indices".

3. Declaration & Initialization of Single Dimensional Array:

Syntax2: Create 1-D array with Size and Initial values

datatype arrayName [size] = {value1, value2, ...} ;

Example2: Declaration of 1-D Array with Size and Initialization

```
// declare an array with size in [] and initial values.
int a[3] = {200, 100, 300};
```

In the above Syntax1 & Syntax2, the **datatype** specifies the type of values we store in that array and **size** specifies the maximum number of values that can be stored in that array.

Syntax3: Create 1-D array without Size and with Initial values

datatype arrayName [ ] = {value1, value2, ...} ;

Example3: Declaration of 1-D Array without Size and with Initialization

```
// declare an array with size in [] and initial values.
float salary[ ] = {1000.25, 2500.75, 3200.77, 500.97};
```

4. Accessing Elements of Single Dimensional Array:

The individual elements of an array are identified using the combination of **'arrayName'** and **'indexNumber'.** The Rules to access (to store or to retrieve) of Single or 1-D Array are,

- array name must be followed by an INDEX number of the element to be accessed.
- index value must be enclosed in square braces [ ].

- **index** value of an element in an array is the reference number given to each element at the time of memory allocation.
- index value of a 1-D array starts with zero (0) for the first element and increments by one for each element.
- index value in an array is also called a **subscript** or **indices**.

Syntax to access individual elements of a single dimensional array:

arrayName [indexNumber]

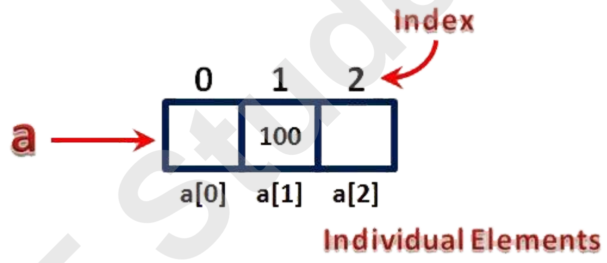
Example: Accessing 1-D array member

```
// declare an array with size 3
int a[3];
```

For this array "a", the individual elements can be denoted as follows. Assigns a value to the 2nd memory location.

a [1] = 100 ;

The result of the above assignment statement is as follows:



Note: The index of an array starts from 0 until it reaches the max (size – 1).

Example 1: [1-D array and "for" loop"]

```
/* Program to Print pre-initialized Array values
#include <stdio.h>
int main()
{   /* Array Declaration and also Initialization */
    int marks[10] = { 90, 91, 99, 93, 94};
    for (int i= 0; i < 5; i++)
    {
       printf("\n Element at position %d is %d",i, marks[i]);
    }
    printf("\n Element at 4th index is %d", marks[3]);
    return 0;
 }
```

OUTPUT:

```
Element at position 0 is 10
Element at position 1 is 91
Element at position 2 is 99
Element at position 3 is 93
Element at position 4 is 94

Element at 4th index is 93
```

5. Example Application 1 of Single Dimensional Array - Sum & Average of n elements

Example 2 : [1-D Array using for loop]

```c
//  Program to Find Sum & Avg of n Elements using Loops and Variables
#include <stdio.h>
int main()
{   int n;
    int sum=0;
    float avg;

    printf("Enter size of the array: ");
    scanf("%d",&n);

    //Declaring array
    int arr[n];
    printf("Enter array elements\n");
     // Input array elements
    for(int i=0;i<n;i++)
    scanf("%d",&arr[i]);

    // Loop to find sum
    for(int i=0;i<n;i++)
          sum+=arr[i];
    printf("\nSum of the array is: %d",sum);
    avg = sum/n;
    printf("\nAverage of the array is: %.2f",avg);
    return 0;
}
```

Output:

```
Enter size of the array: 5
Enter array elements
50
100
75
100
80
Sum of the array is: 405
Average of the array is: 81.00
```

6. Example Application 2 of Single Dimensional Array - Largest of n elements

Example3: [1-D Array in for loop]

```c
// Program to find the largest number in an array using loops
#include <stdio.h>
int main()
{   int size, i, largest;
    printf("\n Enter the size of the array: ");
    scanf("%d", &size);
    int array[size];  //Declaring array
    //Input array elements
    printf("\n Enter %d elements of the array: \n", size);
    for (i = 0; i < size; i++)
    {
        scanf(" %d", &array[i]);
    }
     //Declaring Largest element as the first element
    largest = array[0];
    for (i = 1; i < size; i++)
    {
        if (largest < array[i])
        largest = array[i];
    }
    printf("\n Largest element in the given array is: %d", largest);
    return 0;
 }
```

Output:

```
Enter the size of the array: 3
Enter 3 elements of the array:
90
155
45
Largest element in the given array is: 155
```

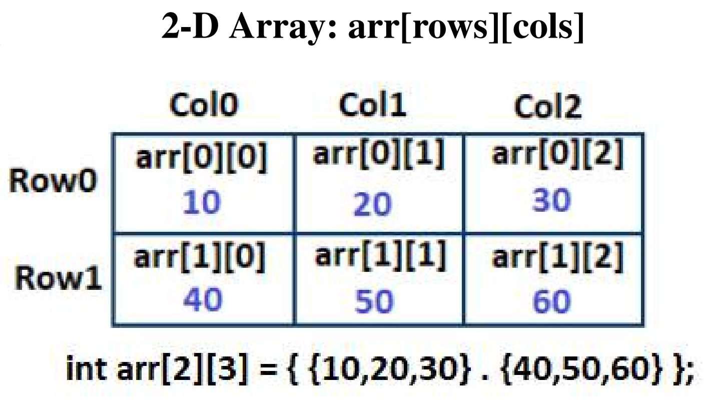
Multi-Dimensional Array

1. Description of Multi Dimensional Array
    a. 2-Dimensional
    b. 3-Dimensional
2. Declaration of Two-Dimensional Array
3. Initialization of Two-Dimensional Array
4. Accessing Elements of Two-Dimensional Array
5. Example Application 1 of Two-Dimensional Array
6. Example Application 2 of Two-Dimensional Array

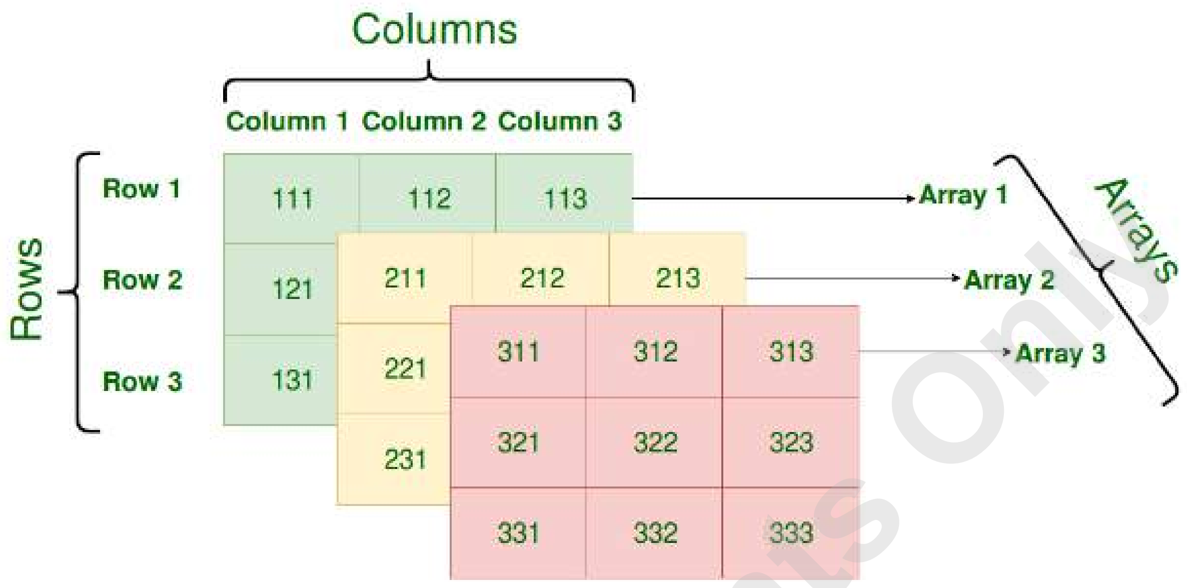
1. Description of Multi-Dimensional Array:

An array of arrays is called a multi-dimensional array. An array created with more than one dimension or size is called a multi-dimensional array.

A multi-dimensional array can be a **two-dimensional array** or **three-dimensional array** or **four-dimensional array** or more.



3-D Array: y[arrays][rows][cols]



y[1][2][1] => 121 y[2][1][2] => 212 y[3][3][3] => 333

The commonly used multi-dimensional array is a **two-dimensional array**. The 2-D arrays are used

- to store data in the form of a table with rows and columns,
- to create mathematical matrices,
- for drawing Chess boards,
- representing structures like a spreadsheet, etc.

2. Declaration of Two-Dimensional Array

Syntax for declaring a two-dimensional array

DataType arrayName [rowIndex] [columnIndex]

Example:

int matrix_A [2][3];

The above declaration of two-dimensional array reserves 12 continuous memory locations of 4 bytes each in the form of **2 rows** and **3 columns**.

3. Initialization of Two-Dimensional Array

Syntax for declaring and initializing a 2-D array with a specific number of rows and columns with initial values.

```
datatype arrayName [rows][colmns] = {
                                      {r1c1 value, r1c2 value, ...},
                                      {r2c1 value, r2c2 value, ...}
                                      …
                                     };
```

Example: Three Methods to Initialize an array (2 x 3 = 6 values)

Method1: First set will be the row1 and next set will be the row2

int matrix_A [2][3] = { {10, 20, 30},{40, 50, 60} };

(or)

```
int matrix_A [2][3] = {
                        {10, 20, 30},
                        {40, 50, 60}
                      };
```

Method2: First 3 values will be the row1 and next 3 values take row2

int matrix_A [2][3] = { 10, 20, 30, 40, 50, 60 };

Method3: User inputs data elements while running the program and saves in the array

```
int matrix_A[2][3];
for(int i = 0; i < 2; i++){
    for(int j = 0; j < 3; j++){
        scanf("%d",&matrix_A[i][j]);
    }
}
```

The above declaration methods of 2-D array reserves 6 contiguous memory locations of 4 bytes each in the form of 2 rows and 3 columns. And the first row is initialized with values 10, 20, 30 and the second row is initialized with values 40, 50, 60.

4. Accessing Individual Elements of Two-Dimensional Array

To access elements of a 2-D array in C, we use the 'arrayName' followed by the [rowIndex] and [columnIndex] of the element that needs to be accessed. Here the row and column index numbers must be enclosed in separate square braces. In the case of the two-dimensional array, the compiler assigns separate index values for rows and columns.

Syntax:

arrayName [rowIndex] [columnIndex]

Example:

matrix_A [0][1] = 10;

In the above statement, the element **10** will be saved at row index 0 and column index 1 of **matrix_A** array.

Note: For **1-D** array, we do not always need to specify the size. But for **2D** array, we must always specify the **column size**.

```
int arr[2][2] = {1, 2, 3,4 }      // Valid declaration
int arr[][2] = {1, 2, 3,4 }       // Valid declaration

// Invalid declaration - column dimension is compulsory
int arr[][] = {1, 2, 3,4 }
// Invalid declaration - column dimension is compulsory
int arr[2][] = {1, 2, 3,4 }
```

5. Example Application 1 of Two-Dimensional Array

```
//Program to print a 2D Array of elements already initialized
#include<stdio.h>
int main(void)
{
    // x array with 3 rows and 2 columns.
    int x[3][2] = {{10,11}, {12,13}, {14,15}};
    // display each array element
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 2; j++)
        {
            printf("Element at x[%i][%i]: ",i, j);
            printf("%d\n",x[i][j]);
        }
    }
     return (0);
}
```

Output:

```
Element at x[0][0]: 10
Element at x[0][1]: 11
Element at x[1][0]: 12
Element at x[1][1]: 13
Element at x[2][0]: 14
Element at x[2][1]: 15
```

6. Example Application 2 of Two-Dimensional Array - Read & Print of m x n size

```
//Program to read and print a 2D Array of m rows and n columns.
#include<stdio.h>
int main()
{//2D: Input number of rows x cols for 3 square arrays
int m,n;
int arr2d[30][30];
printf("\n Enter Number of rows cols for square array: ");
scanf("%d %d",&m,&n);
for(int i=0;i<m;i++)
{
    for(int j=0;j<n;j++)
    {
        printf("Value [%d,%d]: ",i,j);
        scanf("%d",&arr2d[i][j]);
    }
}
//2D: print m x n values
printf("\n 2D array is \n");
for(int i=0; i<m; i++)
{
    for(int j=0;j<n;j++)
    {
        printf(" %d ",arr2d[i][j]);
    }
    printf("\n");
}
return 0;
}
```

OUTPUT: `Enter Number of rows cols for square array:` 2 2

```
Value [0,0]: 10
Value [0,1]: 20
Value [1,0]: 30
Value [1,1]: 40
 2D array is
 10  20
 30  40
```

7. **Example Application 3 of Two-Dimensional Array - Add two 2x2 arrays and save in third array and print the result.**

```
//Arrays 2D: Add two square arrays and svae the result in third array
#include<stdio.h>
int main()
{
//2D: read row1xcol1 values
int rows,cols;
int a1[30][30];
int a2[30][30];
int a3[30][30];
printf("\n Enter rows cols for Arrays a1 and a2: ");
scanf("%d %d",&rows,&cols);

//Read 2D Array1 values
printf("\n Input values for Array1: \n");
```

```
for(int i=0;i<rows;i++)
{
    for(int j=0;j<cols;j++)
    {
        printf("Value [%d,%d]: ",i,j);
        scanf("%d",&a1[i][j]);
    }
}
//Read 2D Array2 values
printf("\n Input values for Array2: \n");
for(int i=0;i<rows;i++)
{
    for(int j=0;j<cols;j++)
    {
        printf("Value [%d,%d]: ",i,j);
        scanf("%d",&a2[i][j]);
    }
}
//2D: print Array1
printf("\n Array1: \n");
for(int i=0;i<rows;i++)
{
    for(int j=0;j<cols;j++)
    {
        printf(" %d ",a1[i][j]);
    }
    printf("\n");
}
//2D: print Array2
printf("\n Array2: \n");
for(int i=0;i<rows;i++)
{
    for(int j=0;j<cols;j++)
    {
        printf(" %d ",a2[i][j]);
    }
    printf("\n");
}
```

```
//2D: Add 2 arrays and save into 3rd array
for(int i=0;i<rows;i++)
{
    for(int j=0;j<cols;j++)
    {
        a3[i][j] = a1[i][j] + a2[i][j];
    }
}
//2D: print Array3 with added values
printf("\n Added values in Array3: \n");
for(int i=0;i<rows;i++)
{
    for(int j=0;j<cols;j++)
    {
        printf(" %d ",a3[i][j]);
    }
    printf("\n");
}
return 0;
}
```

Output:

```
 Enter rows cols for Arrays a1 and a2: 2 2
 Input values for Array1:
Value [0,0]: 10
Value [0,1]: 20
Value [1,0]: 30
Value [1,1]: 40

 Input values for Array2:
Value [0,0]: 1
Value [0,1]: 2
Value [1,0]: 3
Value [1,1]: 4

 Array1:
 10  20
 30  40
```

```
Array2:
1  2
3  4

Added values in Array3:
11  22
33  44
```

Advantages of Array in C

Arrays have a great significance in the C language.

- Arrays make the program optimized and clean
- We can store multiple elements in a single array at once; so, we do not have to write or initialize them multiple times.
- Every element can be traversed in an array using a single loop statement.
- Easier to sort data elements with a few lines of code.
- Any array element can be accessed in any order either from the front or rear in O(1) time.

Applications of Arrays in C

In C, arrays are used in a wide range of applications.

● **Arrays are used to Store List of values -** Single dimensional arrays are used to store a list of values of the same datatype in a row or in a linear form.

● **Arrays are used to Perform Matrix Operations -** Two-dimensional arrays are used to create matrices. We can perform various operations on matrices using two-dimensional arrays.

● **Arrays are used to implement Search Algorithms -** We use single-dimensional arrays to implement search algorithms such as

1. Linear Search
2. Binary Search

● **Arrays are used to implement Sorting Algorithms -** We use Single dimensional arrays to implement sorting algorithms such as,

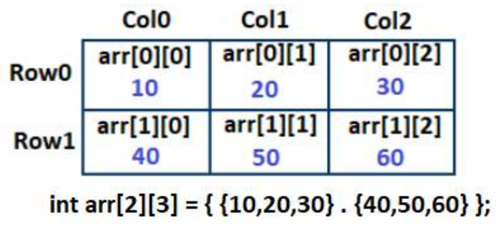
1. Insertion Sort
2. Bubble Sort
3. Selection Sort
4. Quick Sort
5. Merge Sort, etc.,

● **Arrays are used to implement Datastructures -** We use single dimensional arrays to implement data structures such as

1. **Stack Using Arrays**
2. **Queue Using Arrays**

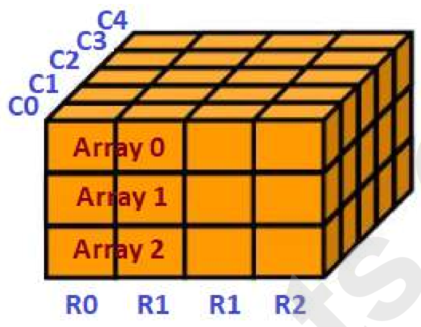
● **Arrays are also used to implement CPU Scheduling Algorithms**

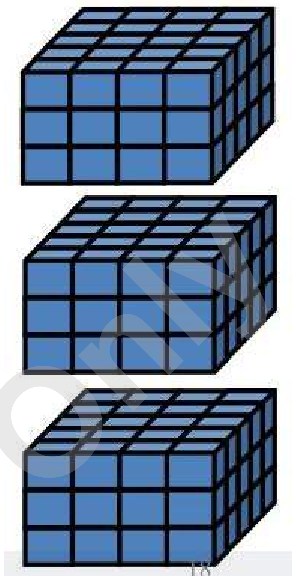
Visual Representation of Single-Dimensional and Multi-Dimensional Arrays



	Col0	Col1	Col2
Row0	arr[0][0] 10	arr[0][1] 20	arr[0][2] 30
Row1	arr[1][0] 40	arr[1][1] 50	arr[1][2] 60

int arr[2][3] = { {10,20,30} . {40,50,60} };



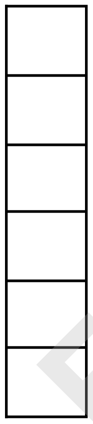


2-D Array
A[3][4]
3 Rows
4 Columns

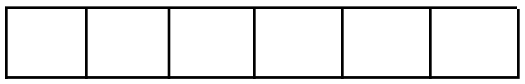
3-D Array
B[3][4][5]
3 Arrays
4 Rows
5 Columns

4-D Array
C[3][4][5][3]
3 Arrays
4 Rows
5 Columns
3 3-D Cubes

1-D Array X[6]



1D Array Y[6]



*** Important Characteristics of Arrays *** [EXAM Bits]

1 An array address is the address of the first element of the array itself.
Ex: **int arr[3] = {100, 200, 300};**
"arr" is the name of the array; it does not refer to any value
"arr" points to the memory address of 1st element
"arr" refers to the address that is same of "&arr[0]"

Ex:
```
#include<stdio.h>
int main(void)
{ int arr[3] = {100, 200, 300};
  printf("Mem Address of array       : %p \n", arr);
  printf("Mem Address of 1st element: %p", &arr[0]);
  return 0;
}
```
Output:
```
Mem Address of array       : 0061FF18
Mem Address of 1st element: 0061FF18
```

2 If you do not initialize an array, you must mention ARRAY SIZE.
Incorrect declaration: **int arr[ ];**
Correct declaration: **int arr[ ] = {5, 15, 25, 35};**
Note: You can skip the SIZE of an array if you initialize with values.

3 Array size is the sum of the sizes of all elements of the array.
Ex: **float salaries[10];** //assuming one float value size is 4 bytes
The total size of the "salaries" array will be: **40bytes** (4bytes x 10 elements)

4 Types of Arrays: int, long, float, double, struct, enum, or char
All elements in one array must be of the same data type.
Ex: **char grade[5] = { 'A', 'B', 'C', 'D', 'F' };**

7. An array's index always starts with 0.

6 An array size can not be changed once it is created.

7 The value (element) in an Array can be changed any number of times.
Ex: **int a[10] = {10, 20, 30};**
a[1] = 15; //this changes the 2nd element 20 to 15.
Now, the array 'a' will have **10, 15, 20** elements

8 To access Nth element of an array "customers", use customers[n-1] because the starting index is 0.

9 arr[i] and i[arr], both notations refer to the same array element.
Ex:
```
char arr[4] = { 'A', 'B', 'C', 'F' };
```

	```
	int i = 0;
	while (i<3) {
	   printf("%c ", arr[i]);
	   printf("%c", i[arr]);
	   printf("\n");
	}
	```
	 Output:
	A A
	B B
	C C
	F F

Note:

For Part-2, Refer to PTC UNIT III, Part-2 document

Part-2:

- **Strings**: String Concepts, C String, String Input / Output Functions, Arrays of Strings, String Manipulation Functions, Programming Example
- **Enumerated, Structure, and Union:** The Type Definition (Type def), Enumerated Types, Structure, Unions, and Programming Application.

PTC UNIT - III, Part-2 (Strings; Struct-union-enum-typedef)

Part-1:

- **Arrays:** Concepts, Using Array in C, Array Application, Two Dimensional Arrays, Multidimensional Arrays, Programming Example

Part-2:

- **Strings**: String Concepts, C String, String Input / Output Functions, Arrays of Strings, String Manipulation Functions, Programming Example
- **Enumerated, Structure, and Union:** The Enumerated Types, Structure, Unions, Type Definition (Type def) and Programming Application.

STRINGS

A. What is a string in C?

A C string is a variable-length array of characters that is delimited by a \0 (null) character.

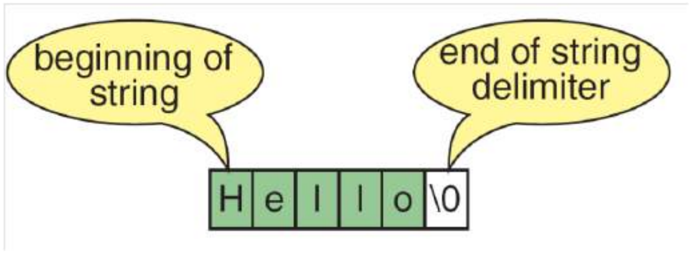
1. **A string is a sequence of characters in C,**
2. **Every string is enclosed within double quotes " ",**
3. **The C compiler automatically adds a '\0' (null) character at the end of a string,**
4. **Strings are created using a one-dimensional array of 'char' datatype.**
5. **Empty String:** A space in " " create an **empty string** by simply adding a \0 (null) character to it.
6. **Size:** the **size** of a string = Number of characters in the string + 1

Note: \0 (null) character takes up 1-byte space at the end of a string. Hence, we must consider one extra space while declaring the size of a string.

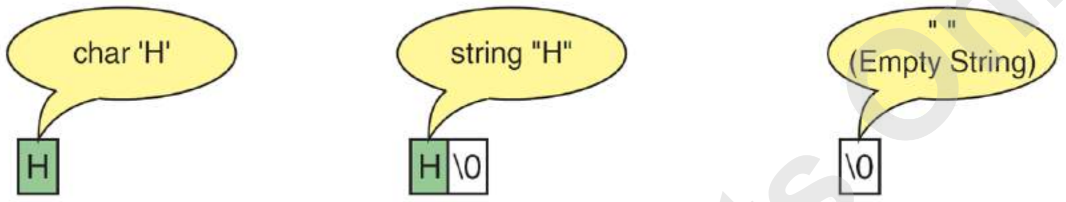
String vs Array of characters:

1. **String:** A **string** must be enclosed in **" "** and the \0 (null) character is automatically added at the end.
2. **Array:** Characters in ' ' create a character array **with NO** null character at the end. That is *NOT a string*!

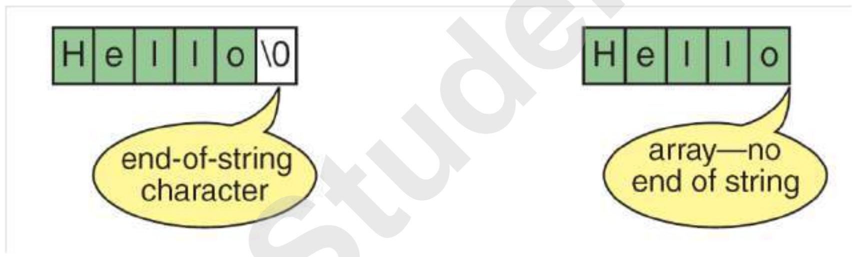
Storing a string "Hello"



Storing Strings and Characters



String vs Character Array



String Constants or Literals:

A series of characters enclosed in **double quotes " "** are called **string constants**. The compiler automatically **adds a \0 (null) character** at the end of each string. The string constants are also called **string literals**.

Ex: char str[9] = "Software"   OR

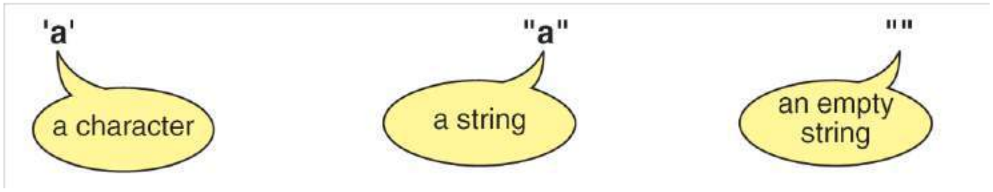
Ex: char str[9] = 'S','o','f','t','w','a','r','e','\0'

Array of Character Constants or Literals:

One or a series of characters enclosed in **single quotes ' '** is called a character constant. It DOES NOT add a \0 (null) character at the end. Hence, it is called an array of characters; not a string.

Ex: 'S','o','f','t','w','a','r','e'

String Literals vs Character Literals



We can access a string literal using its index number as follows:

```
//Access String Literal using its
index number
#include<stdio.h>
int main(){
   printf("%c", "India"[1]);
   return 0;
}
```

"India"[0]	I
"India"[1]	n
"India"[2]	d
"India"[3]	i
"India"[4]	a
"India"[5]	\0

Output:

```
n
```

B. Declaration of Strings in C:

There are 3 methods to declare and create strings in C.

1. **1-D String array** of character datatype (static memory allocation)
2. **2-D Strings array** character datatype (static memory allocation)
3. **Pointer Array** of character datatype (dynamic memory allocation)

[Note: Method 1 & 2 will be explained here. Method 3 will be explained later in Pointers section]

Declaration of Strings in a 1-D array:

In C, strings are created as a one-dimensional array of character datatype. When we create a string, the size of the array must be **1** more than the actual number of characters to be stored. The **1** extra memory block is used to store the END of the string character '\0' (null).

Syntax: Declaration

dataType StringName[size];

dataType is usually 'char',
StringName is any name given to the string variable,

size is the length of the string, i.e the number of characters stored in the string.
Ex: **char str[9];**

str[0]	str[1]	str[2]	str[3]	str[4]	str[5]	str[6]	str[7]	str[8]

String Declaration in C

C. Initializing or Assigning a String value <u>during Declaration of variable</u>

A string value can be initialized or assigned in 5 ways.

1. **Assigning a string without size**

```
char str1[ ] = "Computer";
```

2. **Assigning a string with size**

```
char str2[9] = "Computer";
```

3. **Assigning character by character without size**

```
char str3[ ] = ‘C’,'o,,'m','p','u','t','e','r','\0';
```

4. **Assigning character by character with size**

```
char str4[9] = ‘C’,'o,,'m','p','u','t','e','r','\0';
```

5. **Pointer String**

```
char *ptr = ”Computer”;
```

Example

```
//String Declaration & Initialization
//5 ways to Initialize during declaration
#include<stdio.h>
int main()
{//--- 5 String Initializations during declaration ---
 //String in double quotes
 //Automatically appends NULL character \0 at the end
 char str1[] = "Computer";
 char str2[9] = "Computer";

 //String in an array of chars in single quotes,
 //we MUST Manually append NULL character \0 at the end
 char str3[] = {'C','o','m','p','u','t','e','r','\0'};
 char str4[9] = {'C','o','m','p','u','t','e','r','\0'};
 printf("\n %s",str4);

 //Pointer string
 char *ptr = "Computer";
 printf("\n %s",ptr);
 return 0;
}
```

Ex: `char str[9] = "Computer";`

```
Str        - name of the string
9          - number of characters in the string;
"Computer" - the value of the string
0-8        - are index numbers
```

str[0]	str[1]	str[2]	str[3]	str[4]	str[5]	str[6]	str[7]	str[8]
C	o	m	p	u	t	e	r	\0

String Initialization in C

6. **Assigning or Initialization a string value AFTER Declaration**

Arrays and strings do not support the assignment operator after they are declared.

```
char s1[9];
s1 = "Computer"; // ERROR! array type is not assignable.
```

So, we must use the **strcpy()** function in **string.h** to copy a string value into a variable.

```
strcpy(s1, "Computer");  //copies "computer" string into s1 variable
```

Example:

```
//Assign string value by using strcpy(dest, source) function
//You must include string.h header file
#include<stdio.h>
#include<string.h>
int main()
{//Declaration
 char s1[9];
 //Assigning or Initialization of string values
 strcpy(s1, "Computer");  //using function from string.h header file
 printf("\n %s",s1);
 return 0;
}
```

Output:

```
Computer
```

Here, “**strcpy**” function copies “**Compuer**” string into “**s1**” string variable.

Note:

Strings do not need to be printed character by character like in an array.
Strings can be printed using “printf” statement using “%s” format specifier.

D. Accessing string value (Formatted & Unformatted Input/Output functions in strings)

We can read or print strings in either formatted or unformatted methods.

[formatted input/output]

1. **scanf()** - reads single word using **%s** format; **printf()** - prints a string using **%s** format

[unformatted input/output]

2. **gets()** - reads a line of text from **stdin** until new line, ex: <enter> key
3. **fgets()** - reads a line of text from **stdin** until new line, ex: <enter> key, Or reads a line of text from a file until EOF (end of file)
4. **puts()** - prints a line of text or string to output stream **stdout** without the null character and appends new line (**\n**) character.

1. scanf()

- Using scanf() method we can read **only one word** of the string.
- We use **%s** format specifier to represent a string in “**scanf()**” and “**printf()**” methods.
- **No &** is required before string variable in “**scanf()**”, Ex:
  - **myName[ ]** is a string array of characters;
  - “**myName**” without **‘[’** and **‘]’** will gives the base address of the string variable;
  - No need to use **&** before **myName** while inputting string values.

Example: Input string value using scanf()

```
#include<stdio.h>
int main(){
   char myName[30];
   printf("\n Enter your name : ");
  scanf("%s", myName);
```

```
   printf("Hi! %s, Welcome to Software!", myName);
   return 0;     }
```

2. gets()

- We use a **gets() to** read multiple words or a line of text,
- **Enter** character terminates the **stdin** (input of text from keyboard),
- No need to use **'&'** before string variable name.

Syntax: gets(varName);

varName is the name of the string variable where the string will be saved.

Example: Input string using gets() & Output string using puts()

```
#include<stdio.h>
int main(){
   char myName[50];
   printf("\n Enter your name : ");
   gets(myName);
   printf("Hi! ");
   puts(myName);

   return 0;
}
```

3. fgets()

- We use a **fgets() to** read a line of string from **stdin** or a **file.**
- **Enter** key terminates the **stdin** (input of text from keyboard),
- **EOF** terminates reading from a file,
- No need to use **'&'** before string variable name.

Syntax:

char *fgets(char *str, int n, stdin or FILE *stream);

str is the name of the string variable where the string will be saved.

n - maximum number of characters

stdin - a source from the keyboard (Or)
FILE *stream - a source from file

Example: Read string using fgets() & Output string using puts()

```
//Input a line of text using fgets()
#include <stdio.h>
#define MAX 100
int main()
{
    char str[MAX];
    //Input string from Keyboard (stdin) using "fgets"
    fgets(str, MAX, stdin);

    printf("Your string is: \n");
    // Output string to console using "puts"
    puts(str);

    return 0;
}
```

Note:

- **gets() function is removed from the C standard** because it allows you to input any length of characters. Hence, there might be a buffer overflow. So, **use fgets()** function.
- **fgets()** is the preferred method compared to gets() to input string values. fgets() function allows specifying buffer size and input more than the buffer size.

4. puts( )

Syntax: int puts(const char str[ ]);

- str is the string variable's name.
- Writes a string to **stdout** without the \0 (null) character.
- A newline character (\n) is appended to the output.
- On success a positive value is returned.
- On error EOF is returned.

2-D Array of Strings

1. **Description:** Storing 2 or more strings in an array is called a 2-D Array of Strings.

2. **Declaration:**
   char arrayName[n][m];

   char: Type of values stored in the array
   arrayName: Name of the 2-D array of strings
   n: Max **number of strings** in the array
   m: Max **number of characters** in each string

3. **Initialization:**
   char depts[4][12] = { "Computers", "Electronics", "Electrical", "Civil" };

- 4 strings are saved in the "**depts**" string array
- 10 characters can be saved in each of the 4 strings

Memory	0	1	2	3	4	5	6	7	8	9	10	11
95710	C	o	m	p	u	t	e	r	s	\0		
95720	E	l	e	c	t	r	o	n	i	c	s	\0
95730	E	l	e	c	t	r	i	c	a	l	\0	
95740	C	i	v	i	l	\0						

4. **Access 2-D Array of Strings**

Input
for(i=0 ; i<4 ; i++)
scanf("%s",&name[i][0]);
	The second subscript is **[0]** because,
	● before entering any string the length of the string is 0, and
	● **name[i][0]** points to starting address of each string.

Output
for(i=0 ; i<4 ; i++)
printf("%s \n", name[i]);
	Note: This type of code prints the strings and eliminates any garbage values after '\0'.

5. **Example: Program to search a string in 2-D array of String array**

```c
//Search a string in 2-D array of strings
#include <stdio.h>
#include <string.h>
int main() {
  char names[3][10]; //2-D string array
  char item[10];     //1-D string array to search
  int i, res, status = 0;
  /* Input 3 names */
  printf("Enter 3 names:\n");
  for (i = 0; i < 3; i++)
    scanf("%s", &names[i][0]);
  /*Enter a name to search in the string array*/
  printf("Enter the name to be searched:\n");
  scanf("%s", &item);

  /*Finding the item in the string array*/
  for (i = 0; i < 5; i++)
  { res = strcmp(&names[i][0], item);
    // compares the string in the array with the item and
    // if match is found returns 0 and stores it in variable res
    if (res == 0)
      status = i;
  }
  if (status == 0)  //match is not found
    printf("Given name does not match any name in the list");
  else  ///match is found
    printf("Found. Name in the array exists at index : %d", status);
  return 0;
}
```

Output:

```
Enter 3 names:
Santosh
Vamsi
Abdulla
Enter the name to be searched:
Vamsi
Found. Name in the array exists at index - 1
```

String Library Functions

String Library Functions are predefined functions in C. These functions are used to manipulate string values. They are defined in "**string.h**" header file. We must include **string.h** to use any string handling function.

The following table provides the most commonly used string handling function and their use.

Function	Syntax	Description and Example
strlen()	strlen(str);	Returns length of str. char str[]= "Logic"; printf("Length of string is: %d", strlen(str));

strcpy()	**strcpy(dest, src);**	**The string in src will be copied to dest.**
		`//strcpy() copy a string`
		`#include<stdio.h>`
		`#include<string.h>`
		`int main() {`
		`    char src[] = "Engineer";`
		`    char dest1[10];`
		`    char dest2[10];`
		`    strcpy(dest1, src);`
		`    strcpy(dest2, "Guntur");`
		`printf("%s", dest1); //Engineer`
		`printf("%s", dest2); // Guntur`
		`return 0;   }`
		`Output: Engineer`
		`        Guntur`

strncpy()	strncpy(dest, src, 5)	Copies first 5 characters of src into dest

strcat()	strcat(dest, src)	Appends src string to dest.
		`//strcat() string concatenation`
		`#include<stdio.h>`
		`#include<string.h>`
		`int main()`
		`{  char dest[50] = "Our";`
		`    char src[50] = "  Plan";`
		`    strcat(dest, src);`
		`    printf("%s",dest) ; // Our Plan`
		`    return 0;`
		`}`
		Output: Our Plan

strncat()	strncpy(dest, src, 4)	Appends first 4 characters of src string to dest

strcmp()	strcmp(leftStr, rightStr);	strcmp() compares 2 strings. Checks ASCII value character by character. Returns: 0 if all chars are equal, 1 if a char in 1st string is greater, -1 if a char in 1st string is lesser.

```
#include<stdio.h>
#include<string.h>
int main()
{   char str1[] = "abz";
    char str2[] = "abc";
    int res = strcmp(str1, str2);
    if (res==0)
        printf("Strings are equal");
    else if (res > 0)
        printf("str1 is greater than 
str2");
    else
        printf("str1 is less than str2");
    printf("\nValue returned by strcmp() 
is:  %d" , res);
    return 0;
}
```

Output:

```
str1 is greater than str2
Value returned by strcmp() is:  1
```

strlwr()	strlwr(string1)	Converts all the characters of string1 to lower case.
strupr()	strupr(string1)	Converts all the characters of string1 to upper case.
strrev()	strrev(string1)	It reverses the value of string1.

```
#include<stdio.h>
#include<string.h>
int main()
{   char str1[ ] = "Best Engineer";
 //converts string into uppercase.
    printf("%s\n", strupr(str1));
//converts string into uppercase.
    printf("%s\n", strlwr(str1));
//converts string into uppercase.
    printf("%s\n", strrev(str1));
    return  0;
}
```

Output:

```
BEST ENGINEER
best engineer
reenigne tseb
```

	Other String Functions	
strncmp()	strncmp(string1, string2, 4)	Compares first 4 characters of both string1 and string2
strcmpi()	strcmpi(string1,string2)	Compares two strings, string1 and string2 by ignoring case (upper or lower)
stricmp()	stricmp(string1, string2)	Compares two strings, string1 and string2 by ignoring case (similar to strcmpi())
strdup()	string1 = strdup(string2)	Duplicated value of string2 is assigned to string1
strchr()	strchr(string1, 'b')	Returns a pointer to the first occurrence of character 'b' in string1
strrchr()	'strrchr(string1, 'b')	Returns a pointer to the last occurrence of character 'b' in string1
strstr()	strstr(string1, string2)	Returns a pointer to the first occurrence of string2 in string1
strset()	strset(string1, 'B')	Sets all the characters of string1 to given character 'B'.
strnset()	strnset(string1, 'B', 5)	Sets first 5 characters of string1 to given character 'B'.

atoi()	int atoi(const char *string)	Converts a string to an integer
		Returns the integer value, if successfully.
		1. Returns 0, if the string starts with an alphanumeric character or only contains alphanumeric characters.
		2. Converted to Integer, if string starts with a numeric character & followed by an alphabet. It converts the number to an integer until the occurrence of the first alphabet.
		`#include<stdio.h>`
		`#include <stdlib.h>`
		`int main() {`
		`    char str1[10] = "127";`
		`    char str2[10] = "Namaskar!";`
		`    char str3[10] = "77Hi!";`
		`    char str4[6] = "10.97";`
		`    int x1 = atoi(str1);`
		`    int x2 = atoi(str2);`

		`int x3 = atoi(str3);` `int x4 = atoi(str4);` `printf("Convert'Namaskar!': %d\n", x1);` `printf("Converting '127': %d\n", x2);` `printf("Converting '77Hi!': %d\n", x3);` `printf("Converting '10.97': %d\n", x4);` `return 0;    }`

Example: Program to demonstrate common String Library Functions from string.h

```
//gets(), strlen(), strcpy(), strcat(), strcmp(), strupr(), strlwr()
#include <stdio.h>
#include <string.h>
int main() {
// declaring string variables
char name1[20], name2[30], myname[50];
// Input 2 strings
printf("Enter 1st name:  ");
gets(name1); // input 1st string
printf("Enter 2nd name:  ");
gets(name2); // input 2nd string
// prints the length of the name1[] string
printf("Length of 1st name: %d\n",strlen(name1));

// concatenates the two strings and stores the result in name1[]
printf("Both names are: %s\n", strcat(name1, name2));

// copying the string in name1[] to myname[]
strcpy(myname, name1);
printf("Copied string to myname: %s\n", myname);

// compare the two strings
printf("Compare name1 & name2: %d\n", strcmp(name1, name2));
// convert the string to lowercase
printf("Lower case name1: %s\n",strlwr(name1));
// convert the string to uppercase
printf("Upper case name1: %s\n", strupr(name1));
return 0;
}
```

Output:

```
Enter 1st name:  The Program
Enter 2nd name:   Logic
Length of 1st name: 11
Both names are: The Program Logic
Copied string to myname: The Program Logic
Compare name1 & name2: 1
Lower case name1: the program logic
Upper case name1: THE PROGRAM LOGIC
```

```
Example: /*  Program to Reverse a string */
#include <stdio.h>
#include <string.h>
int main()
{
char ptc[100]; // to store input string
char ptcrev[100]; // to store reveresed string
int len; //to save length of the string

printf("\nEnter a string : ");
fgets(ptc, sizeof ptc, stdin);
len=strlen(ptc);
int j=0;
//len-2 will point to last character
//Ex: Hello len=6   but index is 0 1 2 3 4
for(int i=len-2;i>=0;i--)
{
    ptcrev[j] = ptc[i];
    //printf("ptcrev[%d]=%c", j,ptcrev[j]);
    //printf(" ptc[%d]=%c\n", i,ptc[i]);
    j++;
}
ptcrev[j]='\0';
puts("Reversed String: ");
puts(ptcrev);

return 0;
}

/*  Output:
Enter a string : Computer
Reversed String:
retupmoC
*/
```

Enumerated, Structure, and Union: The Enumerated Types, Structure, Unions, Type Definition (Type def), and Programming Application.

What are User-Defined Data Types in C?

- The User-Defined Data Type is derived from any existing data type in C.
- We can use them for extending the pre-defined data types that are already available in C.
- We can also create various customized data types of your own.

Why do we Need User-Defined Data Types in C?

- The Pre-defined data types (int, char, cfloat etc) and Derived data types (arrays) in C may not offer a wide variety of functions.
- The User-Defined Data Types in C help us define custom data types of our own based on our needs. These data types offer various functions on the basis of how one define them. Hence, these are termed as "User-Defined".

The user-defined data types in C are

1. **enum** - Enumerated data type
2. **struct** - Structure data type
3. **union** - Union data type
4. **typedef** - Type Definition

1. Enumerated Types (enum) in C

Enumeration is the process of creating user defined datatype by assigning names to integral constants

- **enum** is used to create user-defined enumeration datatypes in C.
- The enum data types allow a user to create symbolic names of their own for a list of all integer constants that are related to each other.
- The enumeration helps to set names to integer constants.
- A program becomes more readable by using these names for integer numbers.

Syntax:

enum identifier {name1, name2, name3, ... }

Here, integer 0 will be assigned to name1, integer 1 will be assigned to name2 and so on.

We can also assign our own integral constants as follows.

enum identifier {name1 = 10, name2 = 30, name3 = 15, ... }

Now, the integer 10 will be assigned to name1, the integer 30 will be assigned to name2 and so on.

Example: Program for weekdays using ENUM

```
#include<stdio.h>
#include<conio.h>
enum day { Mon, Tue, Wed, Thu, Fri, Sat, Sun} ;

void main(){
   enum day today;
   today = Tue ;
   printf("\nThis day is %d ", today);
}
```

Output:

```
This day is 1
```

Explanation:

day is a user defined datatype with 7 values as below:

Mon = 0, Tues= 1, Wed= 2, Thu= 3, Fri= 4, Sat= 5, Sun= 6

So when we display **Tue**, the respective integral constant '**1**' will be displayed.

We can also change the order of integral constants, consider the following example program.

Example: Program for weekdays using ENUM with changed integral constant values

```
//Changing order of the integral constants
#include<stdio.h>
#include<conio.h>
enum day { Mon=1, Tue, Wed, Thu, Fri, Sat, Sun} ;

void main(){
   enum day today;
   today = Tue ;
   printf("\nThis day is %d ", today);
}
```

Output:

```
This day is 2
```

Explanation: In this program, the integral constant value starts with '1' instead of '0'. Here, Tuesday's value is displayed as '2'.

We may also define different integral values as we wish.

Example: Program for weekdays using ENUM with different integral constant values

```
//enum with defferent integral constant values
#include<stdio.h>
#include<conio.h>
enum grades {pass = 40, second = 60, first = 80, top = 100} ;
void main(){
   enum grades status;
   status = pass ;
   printf("\nYour are at %d marks. You must study well!", status) ;
}
```

Example: enum declaration

```
enum Boolean { true=1, false=0 };
```

- **enum** is the keyword used to define user defined data type
- **enum** members are basically integers
- Can use expressions like integers
- Makes code easier to read
- Cannot get string equivalent
- Ex: day++ always increments its value by 1
- more than one name can have same integral constant

2. Structures (struct) in C

- A structure is a collection of variables of different data types (non-homogenous) that are related to each other. For example, information of a person, an account, or a part, etc.
- Every data item present in a structure is called as a member. These members are also called fields.
- We use the ***struct*** keyword for creating a structure.

Advantages of Structures:

- Easy to access its members,
- Allocation of all the members is in a continuous memory,
- Faster to access its members

Structure is a colloction of different type of elements under a single name that acts as user defined data type in C.

Syntax Declaration:

```
struct <structure_name>
{
    data_type1 member1;
    data_type2 member2, member3;
    –
   data_type_n member_n;
};
```

- **struct** is a keyword to declare a structure in C.
- **structure_name** is an identifier to use structure
- All members variables must be enclosed in curly braces
- Every structure must be terminated by a semicolon **;**

Example Structure in C:

```
struct Student
{
    char stud_name[30];
    int reg_number;
    float average;
} ;
```

How to create & use a structure variable?

We create a structure variable in two ways.

1. while defining the structure and
2. in main() after terminating structure

How do we access a member of a structure?

To access members of a structure using structure variable, we should use dot (.) operator.

Example: Program to Create and Use structure variables in C

```
#include<stdio.h>
struct Student
{   char stud_name[30];
    int reg_number;
    float average;
    char grade; //F for <40, B for 40-74, A for 75-100
} stud1;
int main(){
    struct Student stud2;  // using struct keyword
    printf("Enter details of stud1 : \n");
    printf("Name : ");
    scanf("%s", stud1.stud_name);
    printf("Roll Number : ");
    scanf("%d", &stud1.reg_number);
    printf("average : ");
    scanf("%f", &stud1.average);

    //Find & set grade
    if(stud1.average<40)
        stud1.grade='F';
    else if(stud1.average>=40 && stud1.average<75)
        stud1.grade='B';
    else
        stud1.grade='A';
    printf("***** Student 1 Details *****\n");
    printf("Student Name        : %s\n", stud1.stud_name);
    printf("Student Reg. Number : %i\n", stud1.reg_number);
    printf("Student Average     : %f\n", stud1.average);
    printf("Student Grade       : %c\n", stud1.grade);
    return 0;    }
```

```
Output:
Enter details of stud1 :
Name : Lokesh
Roll Number : 20450
average : 65

***** Student 1 Details *****
Student Name        : Lokesh
Student Reg. Number : 20450
Student Average     : 65.000000
Student Grade       : B
```

Explanation:

- The stucture variable "stud1" is created while defining the structure
- The structure variable "stud2" is created in main() using struct keyword.
- To access the members of a structure, we use the dot (.) operator.

How the memory is allocated for a structure?

Memory does not get allocated while declaring a structure.
Memory does get allocated when we create the variable of a structure.
Size of memory allocated is equal = Sum of memory required for each member of the structure.

In the above example program, the variables stud_1 and stud_2 are allocated with 36 bytes of memory each.

```
struct Student
{
   char stud_name[30];          30 bytes
   int reg_number;               4 bytes
   float average;                4 bytes
   char grade;                   1 byte
}                         sum = 39 bytes
```

- All the members of a structure can be used simultaneously.
- Until variable of a structure is created no memory is allocated.
- Total memory of a structure variable = Sum of all the memory required by all members of that structure.

3. Unions in C

A union is a collection of elements of different data types (non-homogenus) that are not similar to each other. The union allows the storage of the unrelated elements in the very same memory location. It is also an user-defined data type.

- Unions and Structures are pretty similar to each other
- Difference between Union & Structure is that we can access just a single member of the Union at any given time.
- It is because Union creates memory only for one member that has the biggest size (or the highest number of bytes).
- Elements that are defined in a union are called members of union.
- We use (.) operator to access members of union.

How to create a union?
We declare the union using the "**union**" keyword, and we can access all the members of a Union using the (.) dot operator.

```
union union_name
{  data_type variable_name1;
   data_type variable_name2;
   .
   .
   data_type variable_nameN;
};
```

How to create & use a union variable?
We create a union variable in two ways.

1. while defining the union and
2. in main() after terminating union

How do we access a member of a union?
To access members of a union using union variable, we should use dot (.) operator.

Example: Program to Create and Use union variables in C

```c
#include <stdio.h>
#include <string.h>
union Employee {
   char name[32];
   int age;
   float salary;
};
int main(){
   union Employee employee;
   /* Using one member of a union at a time */
   strcpy(employee.name, "Swathi");
   printf("Name = %s  Address = %p\n", employee.name,
       &employee.name);

   employee.age = 20;
   printf("Age = %d Address = %p\n", employee.age,
       &employee.age);
   employee.salary = 1234.5;
   printf("Salary = %f Address = %p\n", employee.salary,
       &employee.salary);
   /* Printing all member variable of Union, Only last updated
      member will hold it's value remaining will contain garbage */
   printf("\nName = %s\n", employee.name);
   printf("Age = %d\n", employee.age);
   printf("Salary = %f\n", employee.salary);
   printf("\nSize of Union: %d",sizeof employee) ;
   return 0;
}
```

Output:

```
Name = Swathi  Address = 0061FF00
Age = 20 Address = 0061FF00
Salary = 1234.500000 Address = 0061FF00

Name =
Age = 1150963712
Salary = 1234.500000

Size of Union: 32
```

How the memory is allocated for a union?
Memory does not get allocated while declaring a structure.
Memory does get allocated when we create the variable of a structure.
Size of memory allocated is equal = Size of the largest member of the union.

In the above example program, the variables stud_1 and stud_2 are allocated with 36 bytes of memory each.

```
union Employee {
    char name[32];              32 bytes
    int age;                    04 bytes
    float salary;               04 bytes
}emp;                         sum = 32 bytes
```

Here, the emp union variable has been allocated 32 bytes and all members share this memory because same memory exists for all members.

Difference Between Structure and Union in C

Key	Structure	Union
Definition	Structure is the collection of multiple variables of different data types that are related to each other.	Union is also the collection of multiple variables of different data types that are related to each other.
Memory Allocation	In a struct, memory is allocated for all members. All members are accessible in structure.	In the union, the memory is allocated only for its largest member. This single memory is shared by all members. Only one member is accessible in the union at any given time.
Syntax	Declaration of structure in C: **struct struct_name{** **type element1;** **type element2;** **.** **.** **} variable1, variable2, ...;**	Declaration of a union in C: : **union u_name{** **type element1;** **type element2;** **.** **.** **} variable1, variable2, ...;**
Size	Size of Structure = greater or sum of size of all the data members.	Size of union = Size of largest member among all data members.
Value storage	Each member is stored in a separate memory location. Hence, a structure can store separate values for different members.	Union has only one memory allocation of its largest member. All other members share this memory.. So at any given time, the union stores a single value of one of the members.
Initialization	In Structure multiple members can be can be initialized at same time.	However in Union, only the first member can get initialize at a time.

typedef in C

typedef is a keyword used to create alias name for the existing datatypes. Using typedef keyword we can create a temporary name to the primitive data types int, float, char and double.

Syntax:
typedef existing-datatype alias-name

typedef with int data type:

In the following example, Number is defined as alias name for integer datatype. So, we can use Number to declare integer variables.

```
#include<stdio.h>
typedef int Number;
int main(){
   Number a,b,c;    // Here a,b,&c are integer type of variables.
   printf("Enter two integer numbers: ") ;
   scanf("%d%d", &a,&b) ;
   c = a + b;
   printf("Sum = %d", c) ;
}
Output:
Enter two integer numbers: 2 5
Sum = 7
```

typedef with structure or union

```
#include<stdio.h>
typedef struct student
{
    char stud_name[50];
    int stud_rollNo;
}stud;

int main(){
   stud s1;
   printf("Enter the student name: ") ;
   scanf("%s", s1.stud_name);
   printf("Enter the student Roll Number: ");
   scanf("%d", &s1.stud_rollNo);
   printf("\nStudent Information\n");
   printf("Name - %s\nHallticket Number - %d", s1.stud_name,
s1.stud_rollNo);
}
```

```
Output:
Enter the student name: Gopi
Enter the student Roll Number: 553377

Student Information
Name - Gopi
Hallticket Number - 553377
```

Comments: In the above example program, stud is the alias name of student structure. We can use stud as datatype to create variables of student structure. Here, s1 is a variable of student structure datatype.

typedef with Arrays

In C programming language, typedef is also used with arrays. Consider the following example program to understand how typedef is used with arrays.

Example Program to illustrate typedef with arrays in C.

```
#include<stdio.h>
void main(){
   typedef int Items[50];     //Items acts like an integer array type
of size 50
   Items list = {10,20,30,40,50};    //Items is an array of integer
type with size 5.
   int i;
   printf("list elements are : \n") ;
   for(i=0; i<5; i++)
        printf("%d\t", list[i]) ;
}
```

Output:

```
List elements are :
10     20     30     40     50
```

Comments: In this program, Items is the alias name of integer array type of size 50. Here, list is an integer array of size 5.

typedef with Pointers

We can give a name to a pointer data type using typedef. See the following example.

```
#include<stdio.h>
#include<conio.h>
void main(){
   typedef int* intPointer;

   intPointer ptr;  //ptr is a pointer variable of integer datatype.
   int qty = 100;

   ptr = &qty;

   printf("Address of a = %u ",ptr) ;
   printf("\nValue of a = %d ",*ptr);
}
```